### Why Microservices

Traditionally, services were designed as monoliths with all endpoints supported by a single service. A consequence of this approach is that unrelated features of the same product end up in the same server. For example, the service endpoints supporting your product’s reporting needs are not necessarily related to its login or billing endpoints. Yet, they all reside in the same server.

An obvious problem with this architecture is that every minor change in a small part of the code requires deploying the entire code base again. As the product gets bigger, development and deployment become cumbersome with this approach. To make changes and deploy them, developers have to coordinate with every team that works on the server code. This approach is obviously not scalable or agile enough for fast development cycles.

This is where microservices are relevant. In a microservice-based architecture, rather than keeping all the modules together in the same monolithic service, you design separate smaller services for different modules. For example, you can have a small service take care of your billing requirements and another independent service for login. When one module requires information from another, it can just make an API call to the corresponding microservice.

With this decoupling, developers can focus on their particular microservice and deploy it as and when required, rather than waiting for changes from a different module to be finished. Teams can have complete ownership of their entire code base, plus a smaller code base is much easier to maintain. If something goes wrong with a particular microservice, it only brings down that microservice; the rest of the independent microservices will be up and running.

Another advantage is the ability to scale microservices separately. Not all endpoints are used the same amount. For example, the login service is probably used significantly more than, say, billing services are. If we follow a monolith approach, we have to scale up the entire service, thus wasting resources. With a microservice-based approach, we just have to scale up the required microservice. In the above example, we could have a lot more instances of the logging microservice than the billing microservice.

When we decided to move to a microservice architecture at Clover, different designs came up for discussion. One approach we tried was to use Spring Boot along with Spring Cloud. This article discusses the basics of creating a microservice using Spring Boot and Spring Cloud. Spring Boot helps you to create a Spring application with minimal effort, while Spring Cloud provides you with a set of tools that makes communication between different microservices easier.

### ****Spring Boot****

Spring Boot is an efficient framework for creating a Spring-based application. Anyone with basic Java programming skills can quickly create and run a Spring Boot microservice. The learning curve is small, yet it can be used to create robust production-grade applications. Spring Boot and Spring Cloud provide many built-in tools and frameworks that make developing a cloud-based microservice rather easy. Most features can be enabled by just using few annotations, thus making development quite agile.

#### *Creating a simple Spring Boot application*

The easiest way to create a Spring Boot application is to use the [Spring Initializr](https://start.spring.io/). Spring provides you with an easy-to-use interface to generate a basic Spring Boot project with all required dependencies.

You can choose either Maven or Gradle as your build tool. You can select either Java, Kotlin, or Groovy as the programming language, plus your desired Spring Boot version. You can also specify your group and artifact IDs and the dependencies you need.

Spring Initializr will generate a basic Maven/Gradle project that you can import into Eclipse or any other IDE to use as a starting point for your project.

The generated project will have a class (usually named Application) with @SpringBootApplication annotation. This is the starting point of the Spring Boot application. It looks something like this:

The rest of the development is exactly like how you would develop a Spring application. You define your controller layer and use annotations to map requests to methods. The Spring Boot application will pick up annotations and route your requests to the appropriate methods.

Sample request handler:

Use Maven/Gradle builds to generate JARs/WARs to deploy.

#### Deploying your Spring Boot application

* **Embedded servers:**A Spring Boot application can be deployed with an embedded server. You can generate a JAR file and run it like any other JAR file. This avoids having to run and maintain a standalone server instance. Further making this easier, Spring Boot provides support for embedded Jetty, Undertow, and Tomcat, which can be configured in pom.xml (or Gradle), like this:

<dependency>   
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-jetty</artifactId>  
</dependency>

* **Standalone servers:** You can also deploy a Spring Boot application in a standalone server (e.g., standalone Tomcat). For this, you can generate a WAR file for your Spring Boot application (instead of a JAR) and deploy it like a normal WAR file. This will be covered in a separate post.

### ****Spring Cloud****

One of the key challenges in deploying a microservice is handling smooth communication between different microservices. One might require load balancers, some sort of central registry that keeps track of which microservices are up or down, and error handling in case of broken connections, and so on. Fortunately, Spring Cloud provides tools and frameworks to accomplish this easily. Some of the basic Spring Cloud features we tested out are service discovery with Eureka, load balancing with Ribbon, and circuit-breaking with Hystrix.

#### Eureka

One advantage of deploying a microservice is the ability to horizontally scale individual services as and when required. When new instances of a service are spun up dynamically, the other services that use it need to know about the additional availability. Obviously hardcoding addresses of instances of services is not scalable. This is where a service discovery tool comes into play. Service discovery tools allow individual microservices to register themselves with a registry when they are deployed and running. When a client requires a particular service, it can obtain a list of instances of the service from the discovery tool and then query an instance of the service.

Eureka is one such easy to use service discovery tool. The first step in using Eureka is starting an Eureka server. This server acts as the registry to which other services register themselves. When one microservice needs to communicate with another, it can query Eureka to find the active instance of that microservice.

The Eureka server can be another Spring Boot application. Again, you can use [Spring Initializr](https://start.spring.io/) to create this application. Remember to choose Eureka as a dependency in Spring Initializr when you generate your Eureka server application.

After generating the application, edit the application to add annotation @EnableEurekaServer to your Application class (seen in the first gist).

You configure your server port in application.yml. For a local set up, it looks like:

server:   
 port: 8086

eureka:  
 instance:  
 hostname: localhost  
client:  
 registerWithEureka: false  
 fetchRegistry: false  
 serviceUrl:  
 defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/

From the browser, go to http://localhost:port/ to see your Eureka dashboard.

Once you have the Eureka server running, the next step is adding Eureka changes to your microservices so that they register with your Eureka server when deployed. To do this, first add the Eureka dependency to pom.xml:

<dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>  
 <version>2.0.0.RELEASE</version>  
</dependency>

Then make changes in application.yml to add Eureka server configurations. An example for local set up:

eureka:  
 client:  
 registerWithEureka: true  
 fetchRegistry: true  
 serviceUrl:  
 defaultZone: <http://localhost:8086/eureka>  
 instance:  
 hostname: localhost

At the end, add the annotation @EnableDiscoveryClient to your Applicationclass.

Start the service and check the Eureka dashboard. Your service should appear in the list. Optionally, you can start more instances of the service to check that they show up on the dashboard.

#### Ribbon

Once we have Eureka up and running, when your service wants to query another service, it should get a list of instances of the second service from Eureka server and call one of them.

This process is greatly simplified by the use of another tool called Ribbon. Ribbon is a load balancer, which can be configured to automatically obtain a list of instances of a service from Eureka and query the service while balancing the load.

All of your queries to another microservice will be routed through Ribbon, which talks with Eureka to find the actual address of the microservice instance that you need to query. Ribbon can be enabled through application.yml:

ribbon:  
 https:  
 client:  
 enabled: true

Sample code for calling another service using Ribbon and Eureka is given below. In this example, our function calls a second service which is registered with the name DEMOB on Eureka. (Note the serviceUrl: it uses this name and not some address; Ribbon and Eureka take care of resolving the address.)

Also note that we create a RestTemplate bean with @LoadBalanced and @Beanannotations (autowired to the restTemplate protected object in line 1). This is essential to make RestTemplate use Ribbon for load balancing.

#### Hystrix

After setting up your microservices, they obviously need to communicate with each other over the network. Even after your best efforts, API calls to a microservice could fail due to a variety of reasons.

In such situations, you can use Hystrix to provide some level of fault tolerance. Hystrix is a circuit-breaker. Hystrix lets you define a fallback method that gets invoked if your network calls to another microservice fails. It reverts back to normal behavior once the service is available again.

To enable Hystrix, add the following dependencies to pom.xml. The first dependency lets you add Hystrix, while the second dependency lets you enable a Hystrix dashboard, which lets you monitor the health of the service. The actuator dependency is required to collect metrics for Hystrix.

<dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>  
 <version>2.0.0.RELEASE</version>  
</dependency>  
   
<dependency>  
 <groupId>org.springframework.cloud</groupId>\  
 <artifactId>spring-cloud-starter-netflix-hystrix-dashboard</artifactId>  
 <version>2.0.0.RELEASE</version>  
</dependency>  
   
<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-actuator</artifactId>  
</dependency>

To use the Hystrix dashboard, add the following to application.yml:

management:  
 endpoints:  
 web:  
 exposure:  
 include: hystrix.stream

After this, add annotations @EnableHystrixDashboard and @EnableCircuitBreaker to the Application class.

The following code shows how to write a Hystrix wrapper around restTemplate. Inside the callGetService function, if the restTemplate call fails, the control moves to the fallback method mentioned in the @HystrixCommand annotation. (This is the same code used above to illustrate the use of Ribbon.)

### Conclusion

In short, creating a microservice using Spring Boot is rather easy. Most of the work of finding and adding dependencies is taken care of by Spring Initializr. If you are familiar with Spring development, moving to Spring Boot is straightforward. You can enable most of the features using just annotations instead of complex configurations. The time required for learning Spring Boot and building a microservice with it is minimal. Microservices provide an easy way to move away from large monolithic services toward easily maintainable small services, and Spring Boot provides an effective way to build and maintain them.

**Spring Boot Microservices: Building a Microservices Application Using Spring Boot**

Now that we've learned how to set up and run a Spring Boot app using Eclipse IDE and CLI, we'll see how to build a microservices application using Spring Boot.

Join the DZone community and get the full member experience.

[**JOIN FOR FREE**](https://dzone.com/articles/spring-boot-microservices-building-microservices-a)

In a previous pst, we learned how to setup and run [Spring Boot using Eclipse IDE and CLI](https://www.edureka.co/blog/spring-boot-setup-helloworld-microservices-example/). Now in this Spring Boot Microservices post, let me show how we can create Microservices Application for Top Sports Brands using Spring Boot and Netflix Eureka Server in detail. Before creating the application, let me tell you what are the challenges with Microservices Architecture.

Spring Boot enables building production-ready applications quickly and provides non-functional features:

* Embedded servers which are easy to deploy with the containers.
* It helps in monitoring the multiples components.
* It helps in configuring the components externally.

So, let us see the challenges with [microservices architecture](https://dzone.com/articles/microservices-vs-soa-whats-the-difference).

**Challenges with Microservice Architecture**

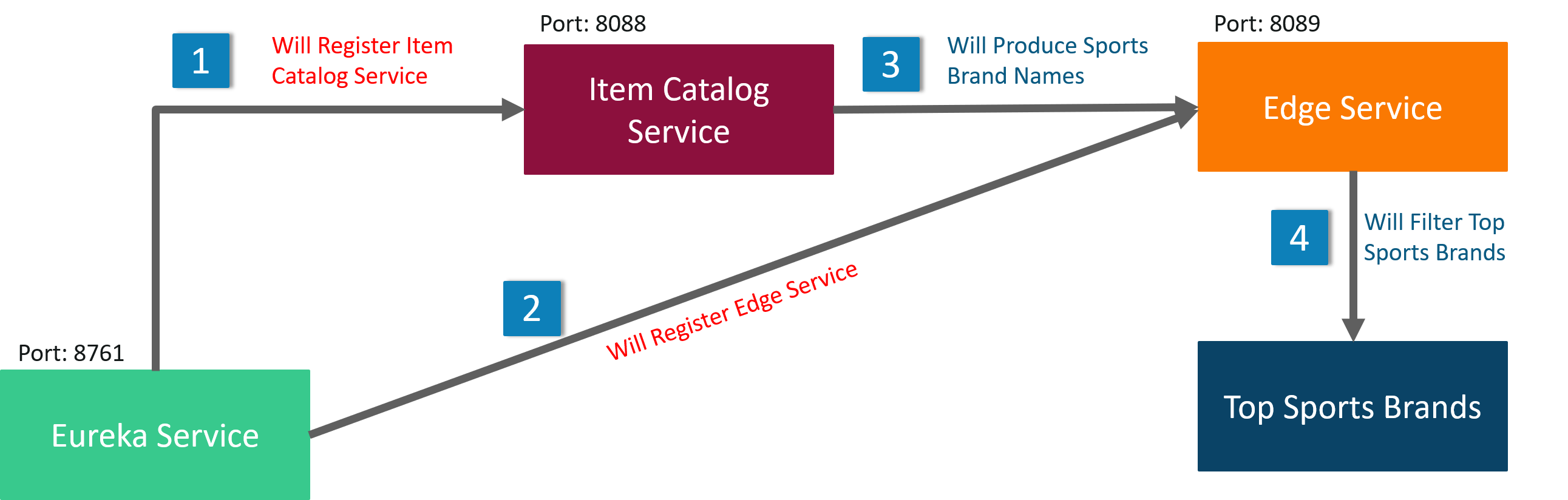
While developing several smaller microservices might look easy, there is several inherent complexities that are associated with microservices architectures. Let's look at some of the challenges:

* **Automating the Components**: It becomes difficult to automate everything because there are several smaller components instead of a monolith, i.e. builds, deployment, monitoring, etc.
* **Perceptibility**: There is several small components to deploy and maintain which sometimes becomes difficult to monitor and identify problems. It requires great perceptibility around all the components.
* **Configuration Management**: There is a great need to maintain the configurations for the components across the various environments.
* **Debugging**: It becomes difficult to probe each service for an error. Centralized Logging and Dashboards are essential to make it easy to debug problems.
* **Consistency**: You cannot have a wide range of tools solving the same problem. While it is important to foster innovation, it is also important to have some decentralized governance around the languages, platforms, technology and tools used for implementing/deploying/monitoring microservices.

**Building Architecture for Top Sports Brands with Spring Boot**

In this Spring Boot microservices example, we will be creating Top Sports Brands' application, which will have three services:

1. **Eureka Service**- This service will register every microservice and then the client microservice will look up the Eureka server to get a dependent microservice to get the job done. This Eureka Server is owned by Netflix and in this, Spring Cloud offers a declarative way to register and invoke services by Java annotation.
2. **Item Catalog Service -**This service will generate the list of sports brands which are popular in the market.
3. **Edge Service -**It is like the standalone Item service created in Bootiful Development with Spring Boot and Angular. However, it will have fallback capabilities which prevent the client from receiving an HTTP error when the service is not available

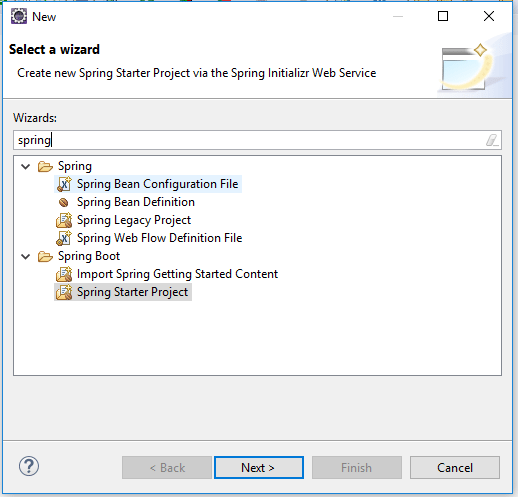


Let us see which of the following tools required to create this Spring Boot microservices example application.

If you are facing any difficulty in installing and running the above tools, please refer to this blog.

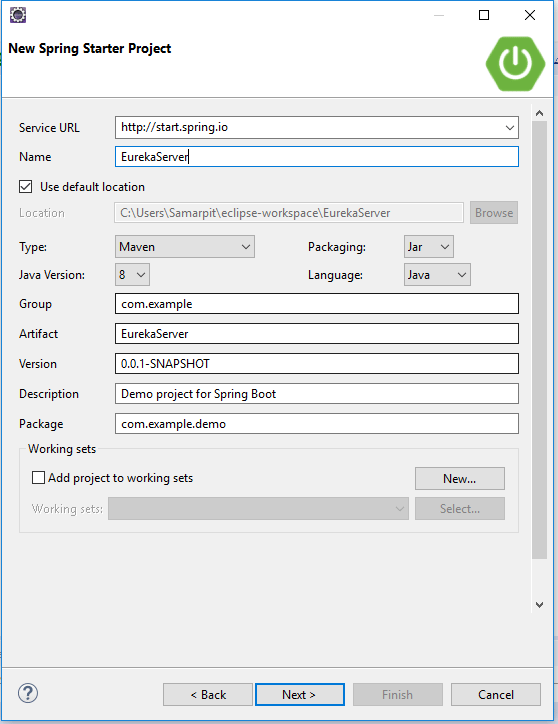
**Creating a Eureka Service**

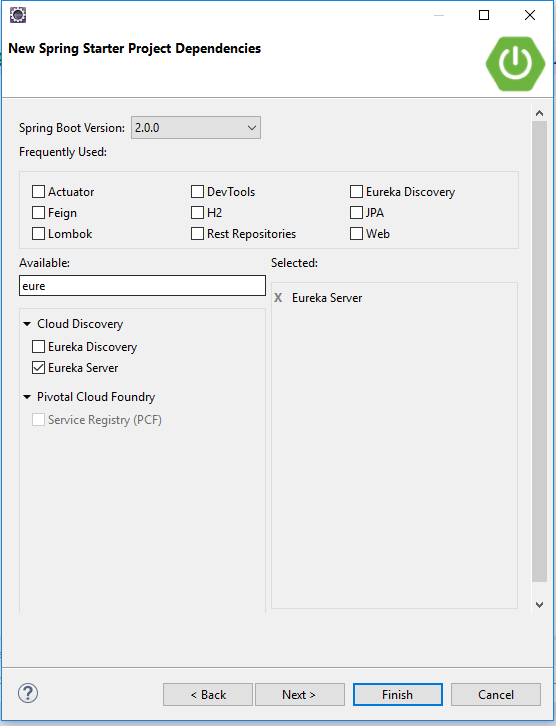
To begin with, create a EurekaServer Spring Starter Project in Eclipse IDE. Click on Spring Starter Project and click on Next.



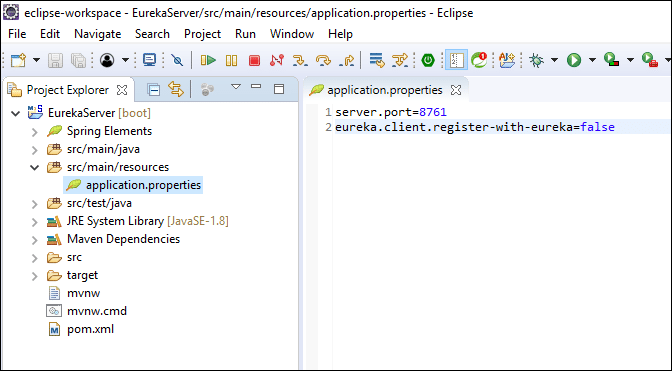
Name your Spring Starter Project as **EurekaServer**and other Information will be filled automatically.

**Note:**Make sure your Internet is connected otherwise it will show an error.





Now, modify EurekaServer/src/main/resources/application.properties file to add a port number and disable registration.

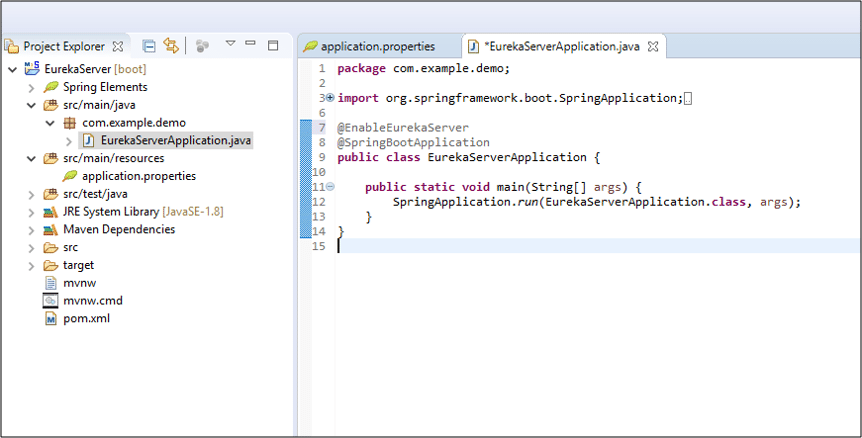


Open EurekaServer/src/main/java/com/example/EurekaServiceApplication.java and add @EnableEurekaServerabove @SpringBootApplication.

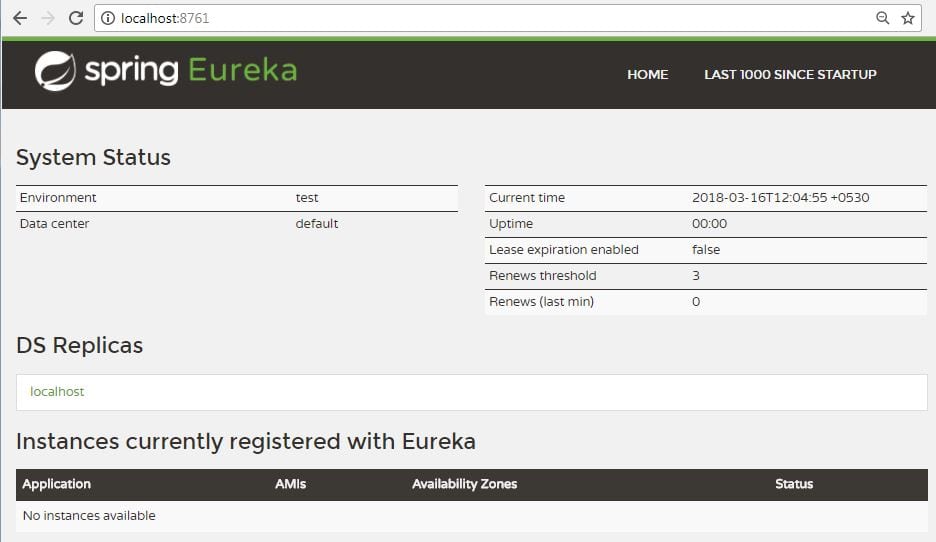
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@EnableEurekaServer @SpringBootApplication

This annotation will configure a registry that will allow other applications to communicate.

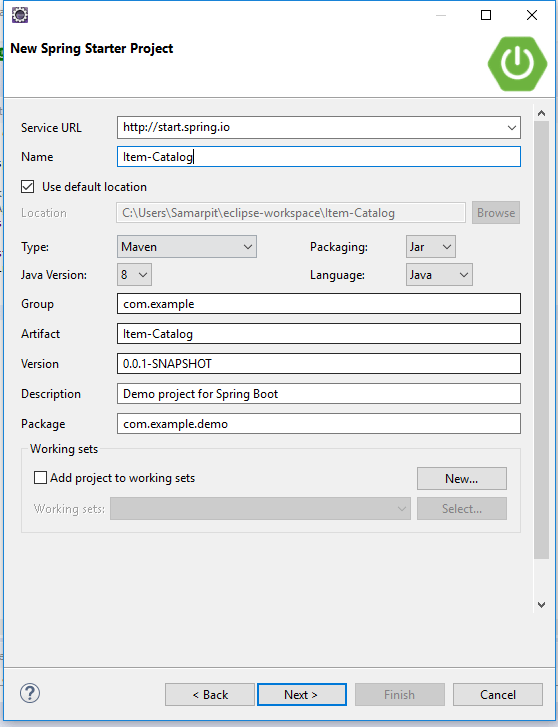
To start the Application: Right Click on theProject -> Run As -> Click on " Spring Boot App "

http://localhost:8761

Now open http://localhost:8761. Here Spring Eureka Server will open and will show no service will be running.

**Spring Boot Microservices: Creating an Item Catalog Service**

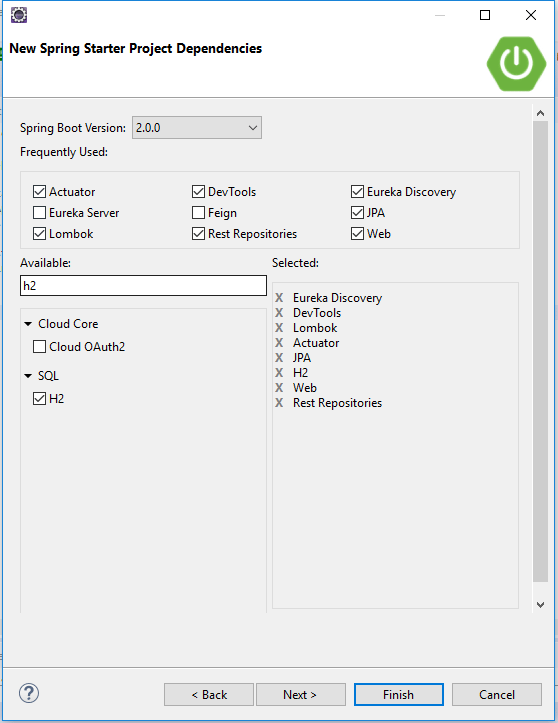
Again create a new project. Use Item-catalog-service for the artifact name and click on **Next**.



Add the following dependencies:

* **Actuator**: features to help you monitor and manage your application
* **EurekaDiscovery**: for service registration
* **JPA**: to save/retrieve data
* **H2**: an in-memory database
* **RestRepositories**: to expose JPA repositories as REST endpoints
* **Web**: Spring MVC and embedded Tomcat
* **DevTools**: to auto-reload the application when files change
* **Lombok**: to reduce boilerplate code

Click on **Finish.**



Now, create an entity, to ItemCatalogServiceApplication.java . The code below assumes you're putting all classes in the same file.

If you're using an editor that doesn't auto-import classes, here's the list of imports needed at the top of ItemCatalogServiceApplication.java.

Add an application name in item-catalog-service/src/main/resources/application.properties file to display in the Eureka service, and set the port to 8088.

Now, Create the Cloud Properties file.

Click on**File -> New -> Other -> File**and add the below code in this file and save it.

eureka.instance.hostname=${vcap.application.uris[0]:localhost}

eureka.instance.nonSecurePort=80

eureka.instance.metadataMap.instanceId=${vcap.application.instance\_id:${spring.application.name}:${spring.application.instance\_id:${server.port}}}

eureka.instance.leaseRenewalIntervalInSeconds = 5

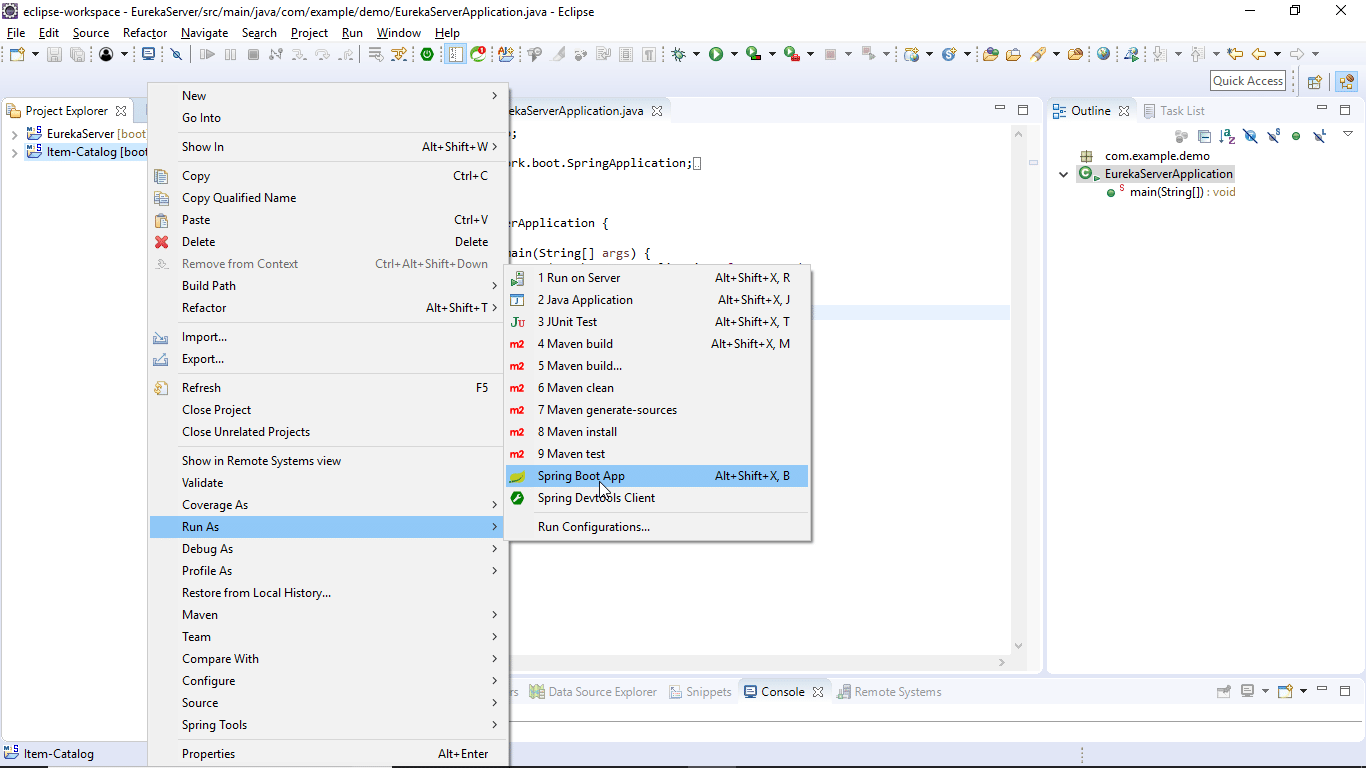
eureka.client.region = default

eureka.client.registryFetchIntervalSeconds = 5

eureka.client.serviceUrl.defaultZone=${vcap.services.pwa-eureka-service.credentials.uri}/eureka/

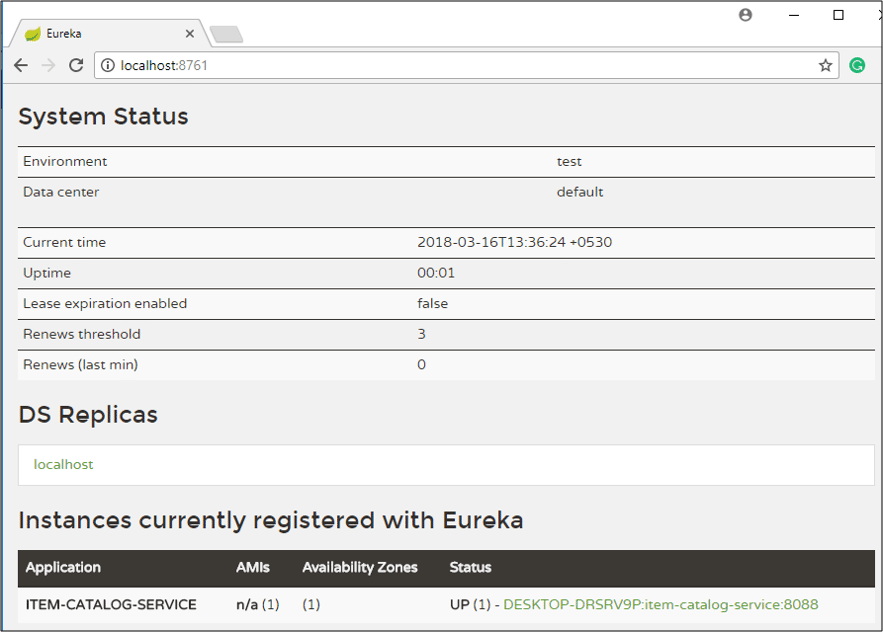
Now, to start the Application:

Right Click on Project -> **Run As** -> Click on " **Spring Boot App** "

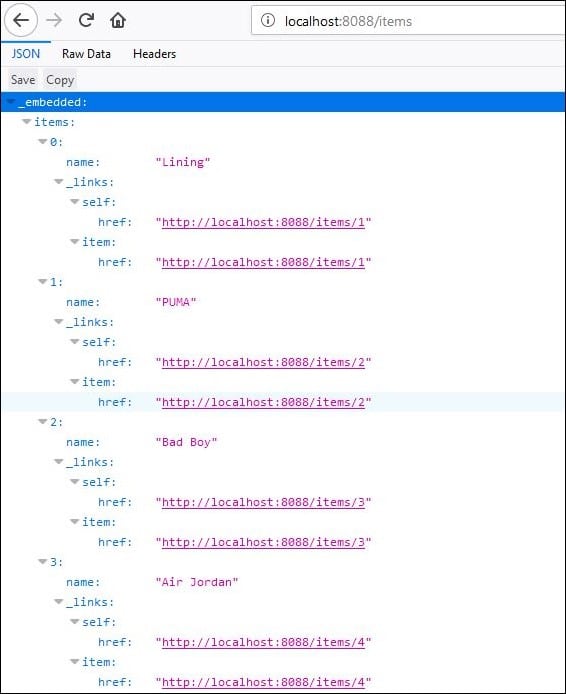


Note: In case of error try this step: Right Click on the **Project** -> **Run As** -> Click on "**Maven Build.**"

Now open http://localhost:8761. Here you will see Item Catalog service will be running.

****

You will see the list of items from the catalog service.

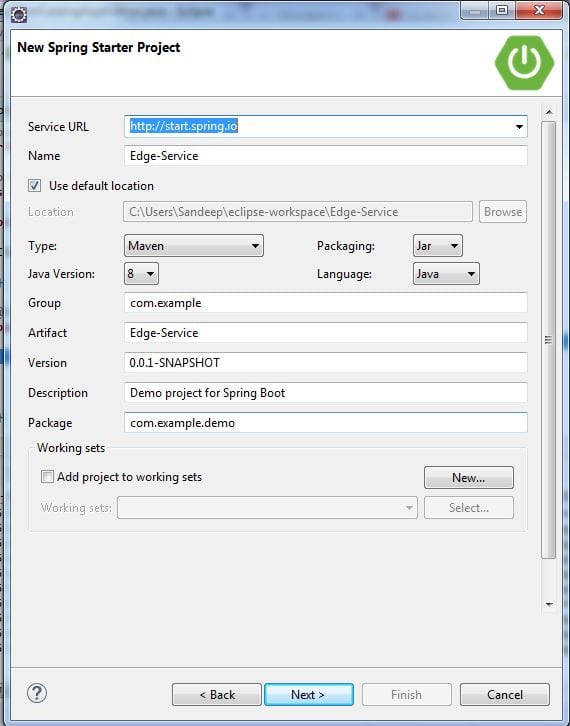


Now let us move forward and create the Edge Service.

**Creating an Edge Service**

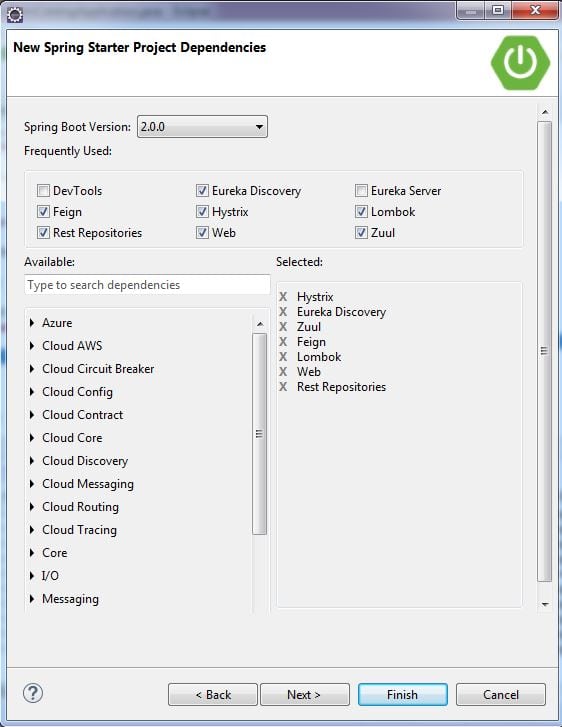
It is similar to the standalone Item service created in **Bootiful Development with Spring Boot and Angular**. However, it will have fallback capabilities which prevent the client from receiving an HTTP error when the service is not available.

Again create a new project. Use edge-service for the artifact name:



* **Eureka Discovery**: for service registration
* **Feign**: a declarative web service client
* **Zuul**: provides intelligent routing
* **Rest Repositories**: to expose JPA repositories as REST endpoints
* **Web**: Spring MVC and embedded Tomcat
* **Hystrix**: a circuit breaker to stop cascading failure and enable resilience
* **Lombok**: to reduce boilerplate code

Click on **Finish.**



Since the item-catalog-service is running on port 8088, you'll need to configure this application to run on a different port. Modify edge-service/src/main/resources/application.properties to set the port to 8089 and set an application name.

Now, Create the Cloud Properties file.

Click on**File -> New -> Other -> File**and add below code in this file and save it.

eureka.instance.hostname=${vcap.application.uris[0]:localhost}

eureka.instance.nonSecurePort=80

eureka.instance.metadataMap.instanceId=${vcap.application.instance\_id:${spring.application.name}:${spring.application.instance\_id:${server.port}}}

eureka.instance.leaseRenewalIntervalInSeconds = 5

eureka.client.region = default

eureka.client.registryFetchIntervalSeconds = 5

eureka.client.serviceUrl.defaultZone=${vcap.services.pwa-eureka-service.credentials.uri}/eureka/

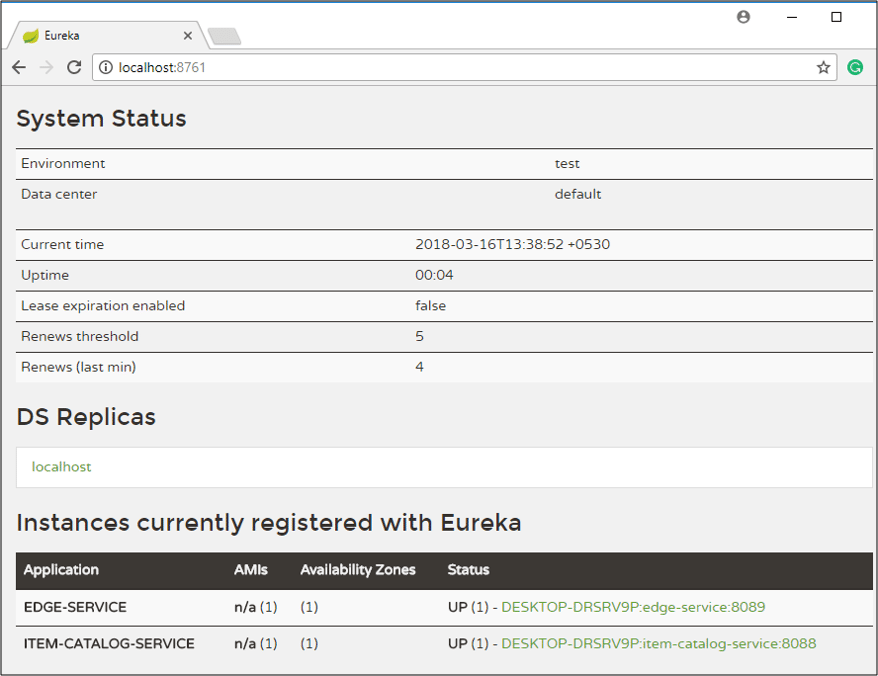
To enable Feign, Hystrix, and registration with the Eureka server, add the appropriate annotations to EdgeServiceApplication.java:

Create a Item DTO (Data Transfer Object) in this same file. Lombok's will generate a methods, getters, setters, and appropriate constructors.

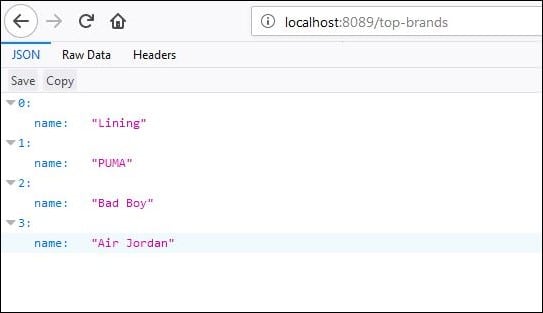
Create a ItemClient interface that uses Feign to communicate to the Item-catalog-service.

Create a RestController below the ItemClient that will filter out less-than-top brands and shows a /top-brands endpoint.

Start the edge-service application with Maven or your IDE and verify it registers successfully with the Eureka server.



Now invoke localhost:8089/top-brands, you will see the list of top brands from the catalog service.



**Note:**If you shut down the item-catalog-service application, you'll get a 500 internal server error.

To fix this, you can use Hystrix to create a fallback method and tell the goodItems() method to use it.

Restart the edge-service and you should see an empty list returned.

Start the item-catalog-service again and this list should eventually return the full list of top brands names.

Microservices implementation example with Spring Boot

Posted by: [Abhimanyu Prasad](https://www.javacodegeeks.com/author/abhimanyu-prasad) in [Software Development](https://www.javacodegeeks.com/category/software-development) December 29th, 2017 [1 Comment](https://www.javacodegeeks.com/2017/12/microservices-implementation-example-spring-boot.html#comments) 6112 Views

1. Introduction

We have already been through the [Microservice demo](https://www.javacodegeeks.com/2017/12/what-are-microservices.html) and it was so pleasing to see the wonderful response it received. Hopefully we are all good with the basics now. This tutorial will guide us all through the actual Microservices implementation, so we are well-directed every time we are asked to create it.

2. Understanding Service Registry

Traditionally, when we consume a REST service, we are usually provided with the network location of the service instance (the REST service URLs that are usually static). However, this tradition has changed with the Microservices architecture coming into picture. Lets understand, how!

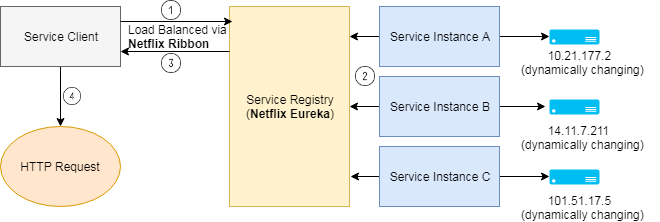
In Microservices architecture, network locations of the service instances are dynamically assigned/changed because of autoscaling, failures and upgrades. To comply with this dynamicity, **service discovery mechanism** comes in.

So there are two main service discovery patterns –

* client‑side discovery
* server‑side discovery

Client‑side discovery

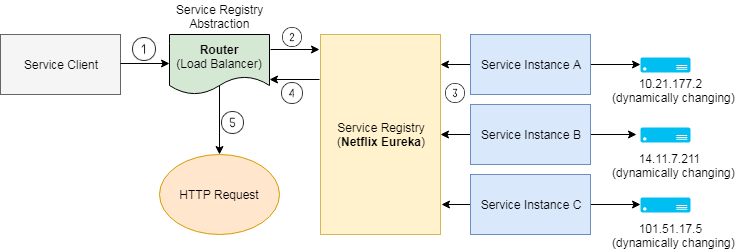
In client‑side service discovery, client queries the Service Registry and determine the available service instances using a separate load balancing mechanism. Load balancing usually works with Service Registry to load balance requests across the available service instances. Each HTTP request is done using one of the load balanced instance that is returned.



One of the major drawback is the tight coupling of client with Service Discovery and the Service discovery logic needs to be written at the client end.

Server‑side discovery

The client queries the Discovery Service via an abstraction layer, which is actually the load balancer or we can say a router, that queries the Service Registry and routes the HTTP request to an available service instance. Details of Service Discovery are abstracted (or hidden) from the client, as a result of which the client don’t need to write the discovery logic, which lead to more ideal loose coupling of client with the Service Discovery.



So what is Service Registry?

**Service registry** is the key part of service discovery mechanism. Service registry is a database of available service instances. Service registry assures that its highly available and up to date with the network locations of the services instances. Clients can cache network locations obtained from the service registry, but as the network locations keep changing, the cache data soon becomes out of date. To cope with this, it is the responsibility of Service Registry server (consists of a cluster of servers) to maintain the consistency and keep refreshing the network location of the service instances (usually refresh is done every 30 seconds).

One of most common examples of a Service Registry is **Netflix Eureka**. It provides a REST API to register (using a POST request) and query (using an HTTP GET request) service instances.

Netflix achieves the high availability by running one or more Eureka servers in each [Amazon EC2 availability zone](http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html). Each Eureka in turn runs on an EC2 instance that has elastic IP address. DNS TEXT records are used to store the Eureka cluster configuration (maps to the list of network locations of Eureka servers), so that when Eureka server starts up, DNS is queried to retrieve the Eureka cluster configuration (network locations of Eureka servers) and assigns itself an unused elastic IP address.

**Hashicorp’s Consul** and **Apache Zookeeper** are other examples of Service Registry.

**Netflix Ribbon** is an IPC (Inter Process Communication) client that works with Eureka to load balance requests across the available service instances. The **@LoadBalanced**annotation configures the RestTemplate to use **Ribbon**, which has been configured to use the Eureka client to query service discovery and fetch available service instances.

3. Microservices Implementation

We will check out a simple demo on Microservices using Spring Boot. Hope we get more close to the Microservices concept.

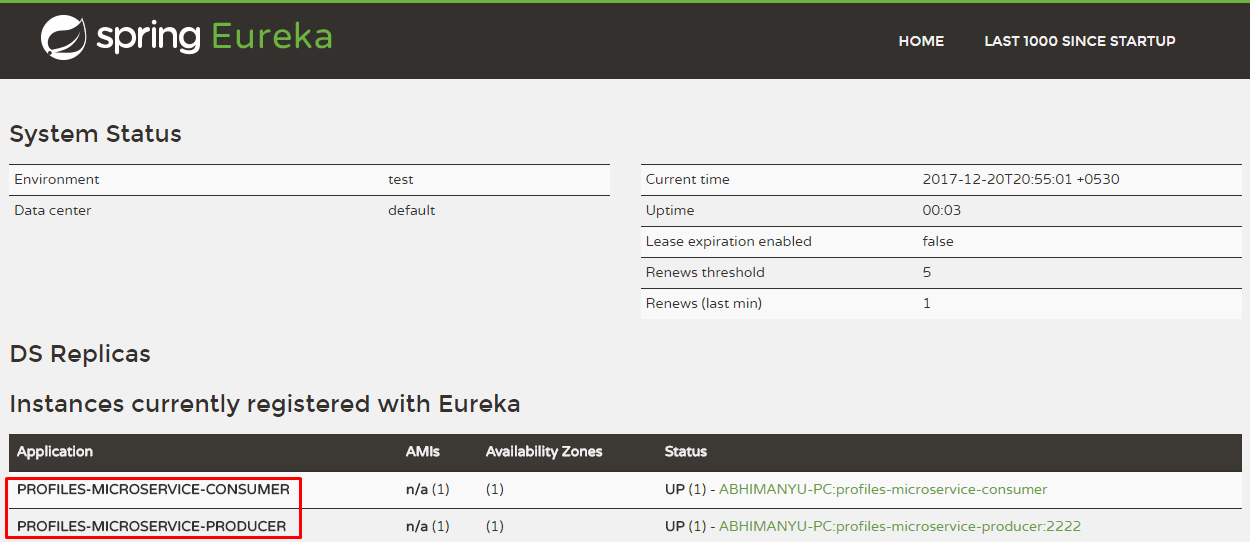
To create a Microservices system, we need to ensure of the below steps –

1. **Creation of Eureka Discovery Server**
   * With Spring Boot, just one annotation **@EnableEurekaServer** does the job.
2. **Creating Producer Microservice**
   * Register itself with Discovery Service
   * **@EnableDiscoveryClient** activates the Netflix Eureka DiscoveryClient implementation
3. **Create Consumer Microservice** which finds the producer service instance registered with Discovery Service
   * Register itself with Discovery Service
   * **@EnableDiscoveryClient** activates the Netflix Eureka DiscoveryClient implementation
   * Requests for DiscoveryClient instance of Producer Microservice using a smart **RestTemplate**.
4. We can then **test end-to-end result** by starting the Eureka service first. Once the Eureka service starts up, start the discovery clients one after the other.

4. Testing the application

Once the Discovery service and Discovery client are started, we can check if the Discovery clients got properly registered themselves with the Eureka Discovery server.

<http://localhost:1111/>

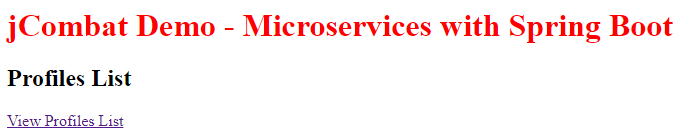


With Spring Boot all the implementation becomes so easy, just the right annotation at the right place with minimal manual configuration.

We will now try running the application as mentioned below –

<http://localhost:8080/>

Project Location: <https://github.com/abhi435/SpringBootMicroservicesDemo>



## ****Top Microservices Interview Questions****

According to Gartner, microservices are the new application platform for cloud development. Microservices are deployed and managed independently, and once implemented inside containers they have very little interaction with the underlying OS. So, if you are planning to start your career in the Microservices and you wish to know the skills related to it, now is the right time to dive in, when the technology is in its nascent state. Hence, to help you prepare for your interviews, I have come up with Microservices Interview Questions and Answers blog.

In this Microservices interview questions blog, I have collected the most frequently asked questions by interviewers. These questions are collected after consulting with [***Microservices Certification Training***](https://www.edureka.co/microservices-architecture-training) experts.

Edureka 2019 Tech Career Guide is out! Hottest job roles, precise learning paths, industry outlook & more in the guide. [***Download***](http://bit.ly/2KmDS3b)now.

In case you have attended any Microservices interview in the recent past, do paste those interview questions in the comments section and we’ll answer them ASAP. You can also comment below if you have any questions in your mind, which you might face in your Microservices interview.

You may go through this recording of Microservices Interview Questions and Answers where our instructor has explained the topics in a detailed manner with examples that will help you to understand this concept better.

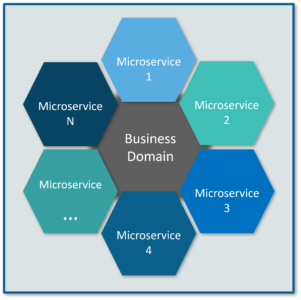
## **Microservices Interview Questions and Answers | Edureka**

### **Q1. List down the advantages of Microservices Architecture.**

|  |  |
| --- | --- |
| **Advantages of Microservices Architecture** | |
| **Advantage** | **Description** |
| **Independent Development** | All microservices can be easily developed based on their individual functionality |
| **Independent Deployment** | Based on their services, they can be individually deployed in any application |
| **Fault Isolation** | Even if one service of the application does not work, the system still continues to function |
| **Mixed Technology Stack** | Different languages and technologies can be used to build different services of the same application |
| **Granular Scaling** | Individual components can scale as per need, there is no need to scale all components together |

### Q2. What do you know about Microservices?

* **Microservices**, aka ***Microservice Architecture***, is an architectural style that structures an application as a collection of small autonomous services, modeled around a **business domain.**
* In layman terms, you must have seen how bees build their honeycomb by aligning hexagonal wax cells.
* They initially start with a small section using various materials and continue to build a large beehive out of it.
* These cells form a pattern resulting in a strong structure which holds together a particular section of the beehive.
* Here, each cell is independent of the other but it is also correlated with the other cells.
* This means that damage to one cell does not damage the other cells, so, bees can reconstruct these cells without impacting the complete beehive.



**Fig 1:** Beehive Representation of Microservices – Microservices Interview Questions

Refer to the above diagram. Here, each hexagonal shape represents an individual service component. Similar to the working of bees, each agile team builds an individual service component with the available frameworks and the chosen technology stack. Just as in a beehive, each service component forms a strong microservice architecture to provide better scalability. Also, issues with each service component can be handled individually by the agile team with no or minimal impact on the entire application.

### **Q3. What are the features of Microservices?**

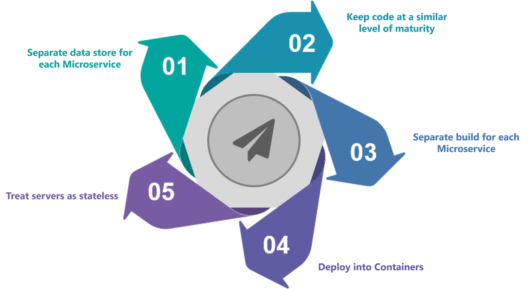


**Fig 3:**Features of Microservices – Microservices Interview Questions

* **Decoupling** – Services within a system are largely decoupled. So the application as a whole can be easily built, altered, and scaled
* **Componentization** – Microservices are treated as independent components that can be easily replaced and upgraded
* **Business Capabilities** – Microservices are very simple and focus on a single capability
* **Autonomy** – Developers and teams can work independently of each other, thus increasing speed
* **Continous Delivery** – Allows frequent releases of software, through systematic automation of software creation, testing, and approval
* **Responsibility** – Microservices do not focus on applications as projects. Instead, they treat applications as products for which they are responsible
* **Decentralized Governance** – The focus is on using the right tool for the right job. That means there is no standardized pattern or any technology pattern. Developers have the freedom to choose the best useful tools to solve their problems
* **Agility** – Microservices support agile development. Any new feature can be quickly developed and discarded again

### **Q4. What are the best practices to design Microservices?**

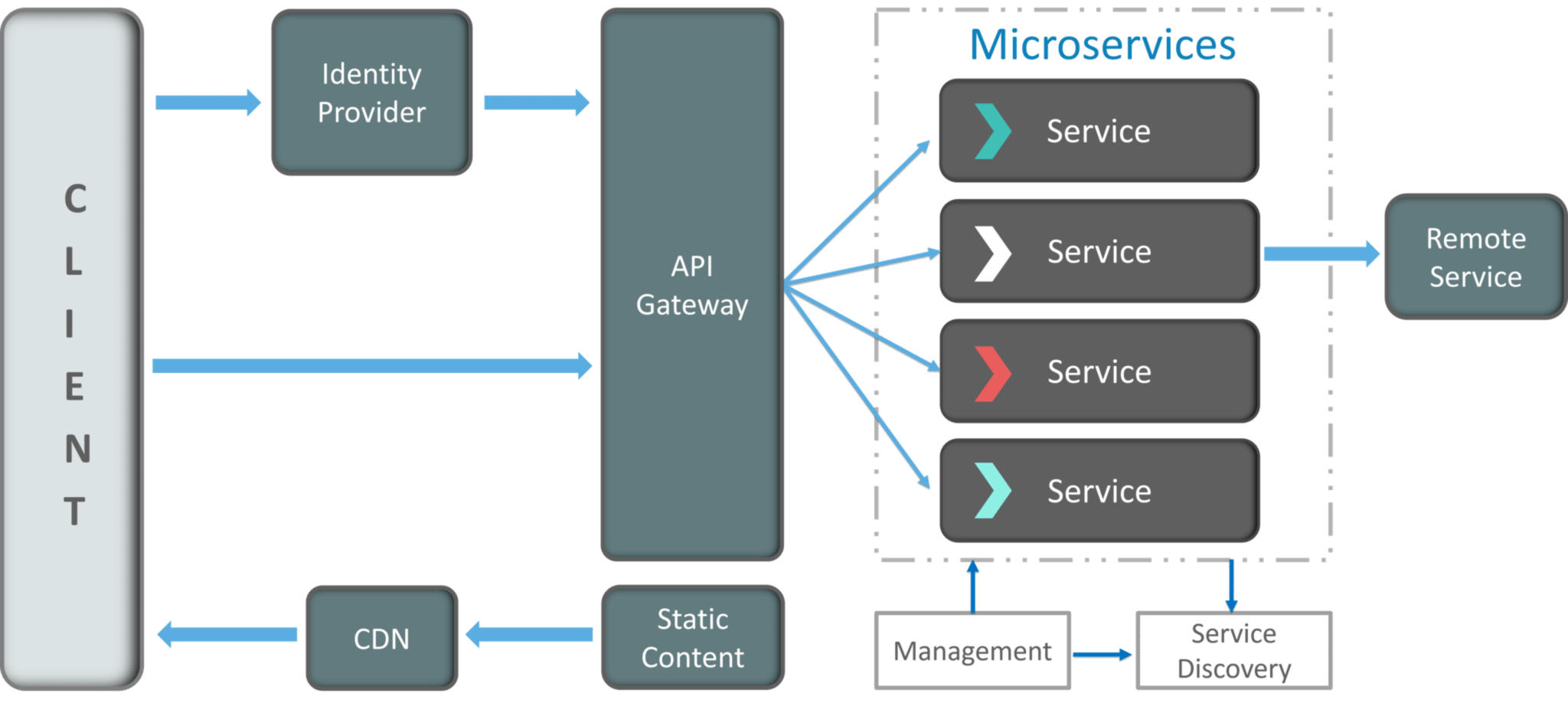
The following are the best practices to design microservices:



**Fig 4:** Best Practices to Design Microservices – Microservices Interview Questions

### **Q5. How does Microservice Architecture work?**

A microservice architecture has the following components:



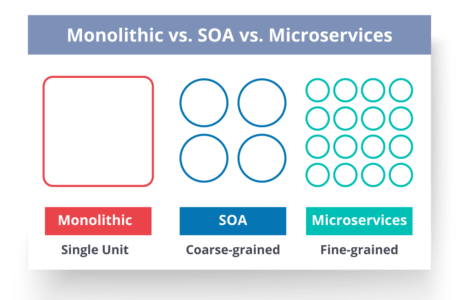
**Fig 5:**Architecture of Microservices – Microservices Interview Questions

* **Clients** – Different users from various devices send requests.
* **Identity Providers** – Authenticates user or clients identities and issues security tokens.
* **API Gateway** – Handles client requests.
* **Static Content** – Houses all the content of the system.
* **Management** –  Balances services on nodes and identifies failures.
* **Service Discovery** – A guide to find the route of communication between microservices.
* **Content Delivery Networks** – Distributed network of proxy servers and their data centers.
* **Remote Service** – Enables the remote access information that resides on a network of IT devices.

### **Q6. What are the pros and cons of Microservice Architecture?**

|  |  |
| --- | --- |
| **Pros of Microservice Architecture** | **Cons of Microservice Architecture** |
| Freedom to use different technologies | Increases troubleshooting challenges |
| Each microservices focuses on single capability | Increases delay due to remote calls |
| Supports individual deployable units | Increased efforts for configuration and other operations |
| Allow frequent software releases | Difficult to maintain transaction safety |
| Ensures security of each service | Tough to track data across various boundaries |
| Mulitple services are parallelly developed and deployed | Difficult to code between services |

### **Q7. What is the difference between Monolithic, SOA and Microservices Architecture?**



**Fig 6:**Comparison Between Monolithic SOA & Microservices – Microservices Interview Questions

* **Monolithic Architecture** is similar to a big container wherein all the software components of an application are assembled together and tightly packaged.
* A **Service-Oriented Architecture** is a collection of services which communicate with each other. The communication can involve either simple data passing or it could involve two or more services coordinating some activity.
* **Microservice Architecture** is an architectural style that structures an application as a collection of small autonomous services, modeled around a business domain.

### **Q8. What are the challenges you face while working Microservice Architectures?**

Developing a number of smaller microservices sounds easy, but the challenges often faced while developing them are as follows.

* **Automate the Components**: Difficult to automate because there are a number of smaller components. So for each component, we have to follow the stages of  Build, Deploy and, Monitor.
* **Perceptibility**: Maintaining a large number of components together becomes difficult to deploy, maintain, monitor and identify problems. It requires great perceptibility around all the components.
* **Configuration Management**: Maintaining the configurations for the components across the various environments becomes tough sometimes.
* **Debugging**: Difficult to find out each and every service for an error. It is essential to maintain centralized logging and dashboards to debug problems.

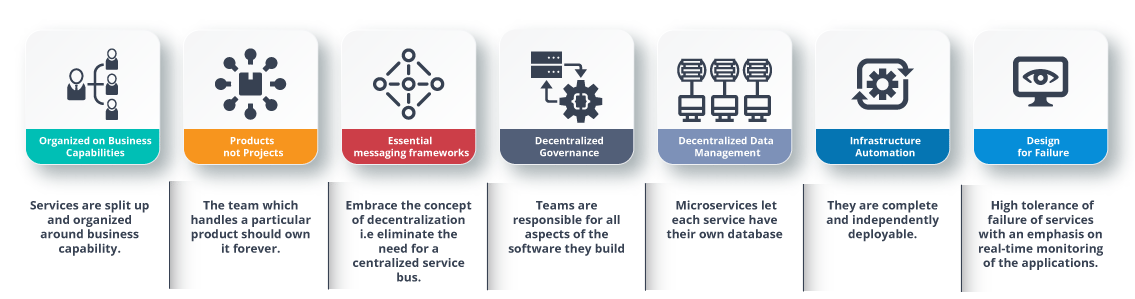
### Q9. What are the key differences between SOA and Microservices Architecture?

The key differences between SOA and microservices are as follows:

|  |  |
| --- | --- |
| **SOA** | **Microservices** |
| Follows “**share-as-much-as-possible**” architecture approach | Follows “**share-as-little-as-possible**” architecture approach |
| Importance is on **business functionality** reuse | Importance is on the concept of “**bounded context**” |
| They have **common** **governance** and standards | They focus on **people** **collaboration** and freedom of other options |
| Uses **Enterprise Service bus (ESB)** for communication | Simple messaging system |
| They support **multiple message protocols** | They use **lightweight protocols** such as **HTTP/REST** etc. |
| **Multi-threaded** with more overheads to handle I/O | **Single-threaded** usually with the use of Event Loop features for non-locking I/O handling |
| Maximizes application service reusability | Focuses on **decoupling** |
| **Traditional Relational Databases** are more often used | **Modern Relational Databases**are more often used |
| A systematic change requires modifying the monolith | A systematic change is to create a new service |
| DevOps / Continuous Delivery is becoming popular, but not yet mainstream | Strong focus on DevOps / Continuous Delivery |

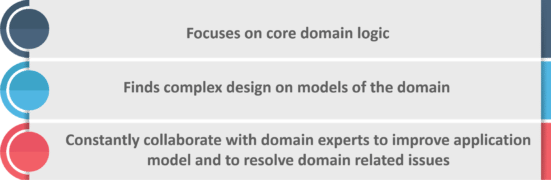
### **Q10. What are the characteristics of Microservices?**

You can list down the characteristics of microservices as follows:



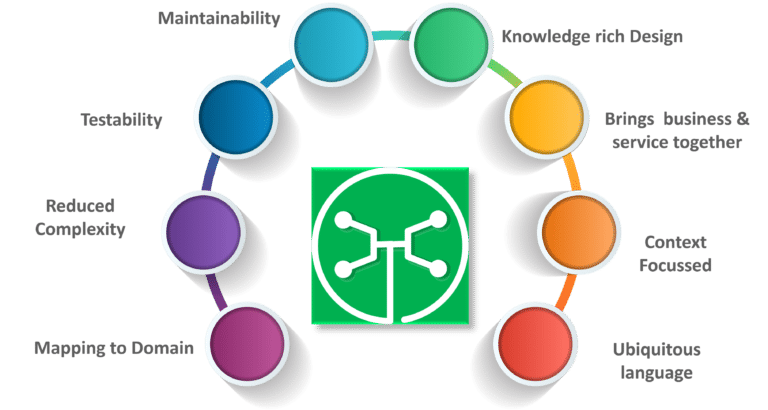
**Fig 7:** Characteristics of Microservices – Microservices Interview Questions

### **Q11. What is Domain Driven Design?**



**Fig 8:**Principles of DDD – Microservices Interview Questions

### **Q12. Why there is a need for Domain Driven Design (DDD)?**



**Fig 9:** Factors Why we need DDD – Microservices Interview Questions

### **Q13. What is Ubiquitous language?**

If you have to define the **Ubiquitous Language (UL)**, then it is a common language used by developers and users of a specific domain through which the domain can be explained easily.

The ubiquitous language has to be crystal clear so that it brings all the team members on the same page and also translates in such a way that a machine can understand.

### **Q14. What is Cohesion?**

The degree to which the elements inside a module belong together is said to be **cohesion**.

[[](https://www.edureka.co/microservices-architecture-training)](https://www.edureka.co/microservices-architecture-training" \t "_blank)

### [Microservices Architecture Training](https://www.edureka.co/microservices-architecture-training" \t "_blank)

* [Instructor-led Sessions](https://www.edureka.co/microservices-architecture-training" \t "_blank)
* [Real-life Case Studies](https://www.edureka.co/microservices-architecture-training" \t "_blank)
* [Assignments](https://www.edureka.co/microservices-architecture-training" \t "_blank)
* [Lifetime Access](https://www.edureka.co/microservices-architecture-training" \t "_blank)

[Explore Curriculum](https://www.edureka.co/microservices-architecture-training" \t "_blank)

### **Q15.  What is Coupling?**

The measure of the strength of the dependencies between components is said to be **coupling**. A good design is always said to have **High Cohesion** and **Low Coupling**.

#### Subscribe to our youtube channel to get new updates..!

### **Q16.  What is REST/RESTful and what are its uses?**

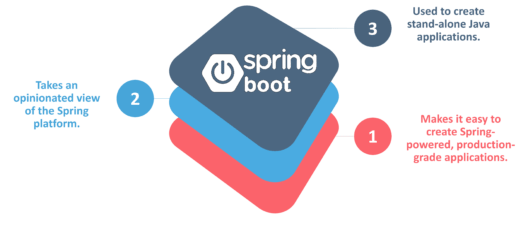
**Representational State Transfer (REST)/RESTful** web services are an architectural style to help computer systems communicate over the internet. This makes microservices easier to understand and implement.

Microservices can be implemented with or without RESTful APIs, but it’s always easier to build loosely coupled microservices using RESTful APIs.

### **Q17. What do you know about Spring Boot?**

It’s a knows fact that spring has become more and more complex as new functionalities have been added. If you have to start a new spring project, then you have to add build path or add maven dependencies, configure application server, add spring configuration. So everything has to be done from scratch.

**Spring Boot** is the solution to this problem. Using spring boot you can avoid all the boilerplate code and configurations. So basically consider yourself as if you’re baking a cake spring is like the ingredients that are required to make the cake and spring boot is the complete cake in your hand.



**Fig 10:**Factors of Spring Boot – Microservices Interview Questions

### **Q18. What is an actuator in Spring boot?**

Spring Boot actuator provides restful web services to access the current state of running an application in the production environment. With the help of actuator, you can check various metrics and monitor your application.

### **Q19. What is Spring Cloud?**

According to the official website of Spring Cloud, Spring Cloud provides tools for developers to quickly build some of the common patterns in distributed systems (e.g. configuration management, service discovery, circuit breakers, intelligent routing, leadership election, distributed sessions, cluster state).

### **Q20. What problems are solved by Spring Cloud?**

While developing distributed microservices with Spring Boot we face few issues which are solved by Spring Cloud.

* **The complexity associated with distributed systems –**This includes network issues, Latency overhead, Bandwidth issues, security issues.
* **Ability to handle Service Discovery –**Service discovery allows processes and services in a cluster to find each other and communicate.
* **Solved redundancy issues –**Redundancy issues often occur in distributed systems.
* **Load balancing –**Improves the distribution of workloads across multiple computing resources, such as a computer cluster, network links, central processing units.
* **Reduces performance issues –**Reduces performance issues due to various operational overheads.

### **Q21.  What is the use of WebMvcTest annotation in Spring MVC applications?**

**WebMvcTest** annotation is used for unit testing Spring MVC Applications in cases where the test objective is to just focus on Spring MVC Components. In the snapshot shown above, we want to launch only the ToTestController. All other controllers and mappings will not be launched when this unit test is executed.

### **Q22. Can you give a gist about Rest and Microservices?**

#### **REST**

Though you can implement microservices in multiple ways, REST over HTTP is a way to implement Microservices. REST is also used in other applications such as web apps, API design, and MVC applications to serve business data.

#### **Microservices**

Microservices is an architecture wherein all the components of the system are put into individual components, which can be built, deployed, and scaled individually. There are certain principles and best practices of Microservices that help in building a resilient application.

In a nutshell, you can say that REST is a medium to build Microservices.

### **Q23. What are different types of Tests for Microservices?**

While working with microservices, testing becomes quite complex as there are multiple microservices working together. So, tests are divided into different levels.

* At the **bottom level**, we have **technology-facing tests** like- unit tests and performance tests. These are completely automated.
* At the **middle level**, we have tests for **exploratory testing** like the stress tests and usability tests.
* At the **top level,**we have **acceptance tests** that are few in number. These acceptance tests help stakeholders in understanding and verifying software features.

### **Q24. What do you understand by Distributed Transaction?**

**Distributed Transaction** is any situation where a single event results in the mutation of two or more separate sources of data which cannot be committed atomically. In the world of microservices, it becomes even more complex as each service is a unit of work and most of the time multiple services have to work together to make a business successful.

### **Q25. What is an Idempotence and where it is used?**

**Idempotence** is the property of being able to do something twice in such a way that the end result will remain the same i.e. as if it had been done once only.

**Usage**: Idempotence is used at the remote service, or data source so that, when it receives the instruction more than once, it only processes the instruction once.

### **Q26. What is Bounded Context?**

Bounded Context is a central pattern in Domain-Driven Design. It is the focus of DDD’s strategic design section which is all about dealing with large models and teams. DDD deals with large models by dividing them into different Bounded Contexts and being explicit about their inter-relationships.

### **Q27. What is Two Factor Authentication?**

Two-factor authentication enables the second level of authentication to an account log-in process.

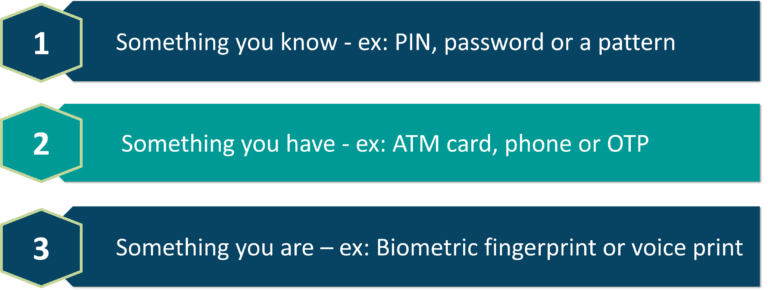


**Fig11:**Representation of Two Factor Authentication – Microservices Interview Questions

So suppose a user has to enter only username and password, then that’s considered a single-factor authentication.

### **Q28. What are the types of credentials of Two Factor Authentication?**

The three types of credentials are:



**Fig 12:**Types of Credentials of Two Factor Authentication – Microservices Interview Questions

### **Q29. What are Client certificates?**

A type of digital certificate that is used by client systems to make authenticated requests to a remote server is known as the **client certificate**. Client certificates play a very important role in many mutual authentication designs, providing strong assurances of a requester’s identity.

### **Q30. What is the use of PACT in Microservices architecture?**

**PACT** is an open source tool to allow testing interactions between service providers and consumers in isolation against the contract made so that the reliability of Microservices integration increases.

#### **Usage in Microservices:**

* Used to implement Consumer Driven Contract in Microservices.
* Tests the consumer-driven contracts between consumer and provider of a Microservice.

### **Q31. What is OAuth?**

**OAuth**stands for open authorization protocol. This allows accessing the resources of the resource owner by enabling the client applications on HTTP services such as third-party providers Facebook, GitHub, etc. So with this, you can share resources stored on one site with another site without using their credentials.

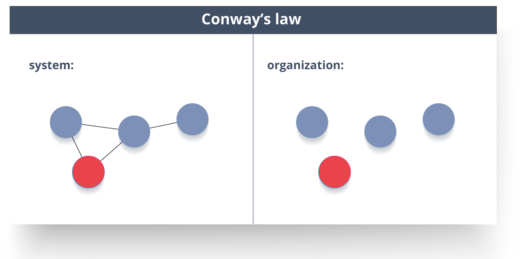
[[](https://www.edureka.co/microservices-architecture-training)](https://www.edureka.co/microservices-architecture-training" \t "_blank)

### [Microservices Architecture Training](https://www.edureka.co/microservices-architecture-training" \t "_blank)

[Watch The Course Preview](https://www.edureka.co/microservices-architecture-training" \t "_blank)

### **Q32. What is Conway’s law?**

“Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization’s communication structure.” –***Mel Conway***



**Fig 13:**Representation of Conway’s Law – Microservices Interview Questions

This law basically tries to convey the fact that, in order for a software module to function, the complete team should communicate well. Therefore the structure of a system reflects the social boundaries of the organization(s) that produced it.

### **Q33. What do you understand by Contract Testing?**

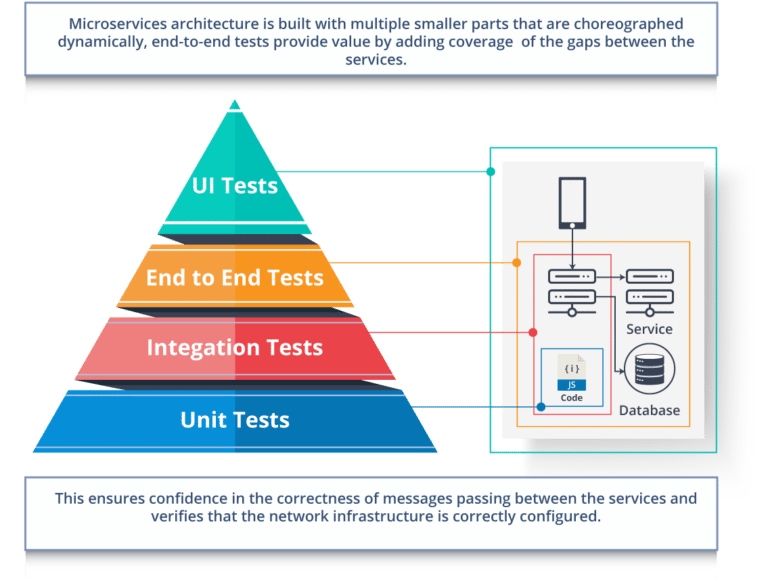
According to Martin Flower, **contract test** is a test at the boundary of an external service which verifies that it meets the contract expected by a consuming service.

Also, contract testing does not test the behavior of the service in depth. Rather, it tests that the inputs & outputs of service calls contain required attributes and the response latency, throughput is within allowed limits.

### **Q34. What is End to End Microservices Testing?**

End-to-end testing validates each and every process in the workflow is functioning properly. This ensures that the system works together as a whole and satisfies all requirements.

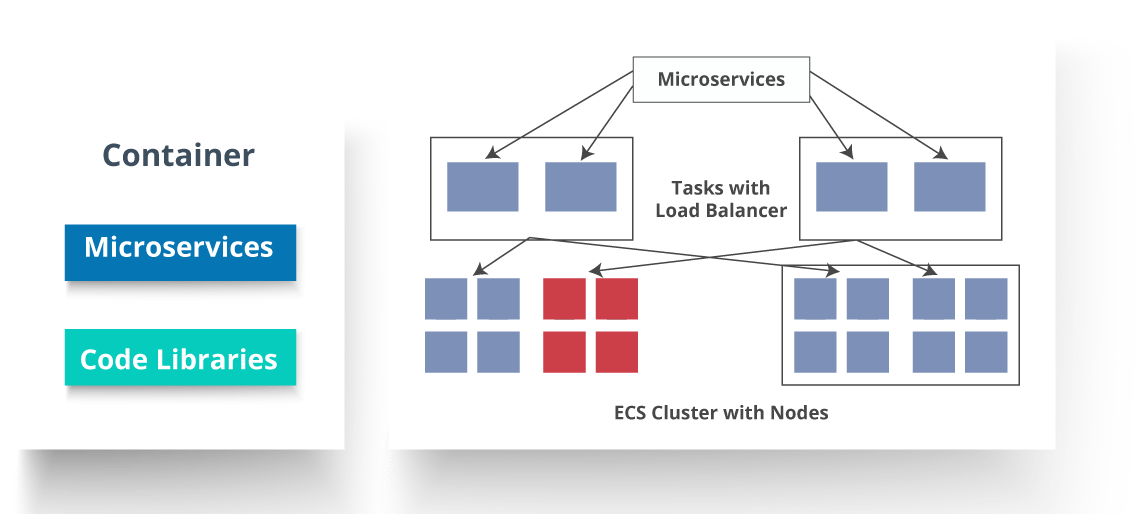
In layman terms, you can say that end to end testing is a kind of tests where everything is tested after a particular period.



**Fig 14:** Hierarchy of Tests – Microservices Interview Questions

### **Q35. What is the use of Container in Microservices?**

Containers are a good way to manage microservice based application to develop and deploy them individually*.* You can encapsulate your microservice in a container image along with its dependencies, which then can be used to roll on-demand instances of microservice without any additional efforts required.



**Fig 15:**Representation of Containers and How they are used in Microservices – Microservices Interview Questions

### **Q36. What is DRY in Microservices architecture?**

**DRY** stands for **Don’t Repeat Yourself**. It basically promotes the concept of reusing the code. This results in developing and sharing the libraries which in turn result in tight coupling.

### **Q37. What is a Consumer-Driven Contract (CDC)?**

This is basically a pattern for developing Microservices so that they can be used by external systems. When we work on microservices, there is a particular provider who builds it and there are one or more consumers who use Microservice.

Generally, providers specify the interfaces in an XML document. But in Consumer Driven Contract, each consumer of service conveys the interface expected from the Provider.

### **Q38**. What is the role of Web, RESTful APIs in Microservices?

A microservice architecture is based on a concept wherein all its services should be able to interact with each other to build a business functionality. So, to achieve this, each microservice must have an interface. This makes the web API a very important enabler of microservices. Being based on the open networking principles of the Web, RESTful APIs provide the most logical model for building interfaces between the various components of a microservice architecture.

### **Q39. What do you understand by Semantic monitoring in Microservices architecture?**

Semantic monitoring, also known as**synthetic monitoring** combines automated tests with monitoring the application in order to detect business failing factors.

### **Q40. How can we perform Cross-Functional testing?**

Cross-functional testing is a verification of non-functional requirements, i.e. those requirements which cannot be implemented like a normal feature.

### **Q41. How can we eradicate non-determinism in tests?**

**Non-Deterministic Tests** (NDT)  are basically unreliable tests.  So, sometimes it may happen that they pass and obviously sometimes they may also fail. As and when they fail, they are made to re-run to pass.

Some ways to remove non-determinism from tests are as follows:

1. Quarantine
2. Asynchronous
3. Remote Services
4. Isolation
5. Time
6. Resource leaks

### **Q42. What is the difference between Mock or Stub?**

#### **Stub**

* A dummy object that helps in running the test.
* Provides fixed behavior under certain conditions which can be hard-coded.
* Any other behavior of the stub is never tested.

For example, for an empty stack, you can create a stub that just returns true for empty() method. So, this does not care whether there is an element in the stack or not.

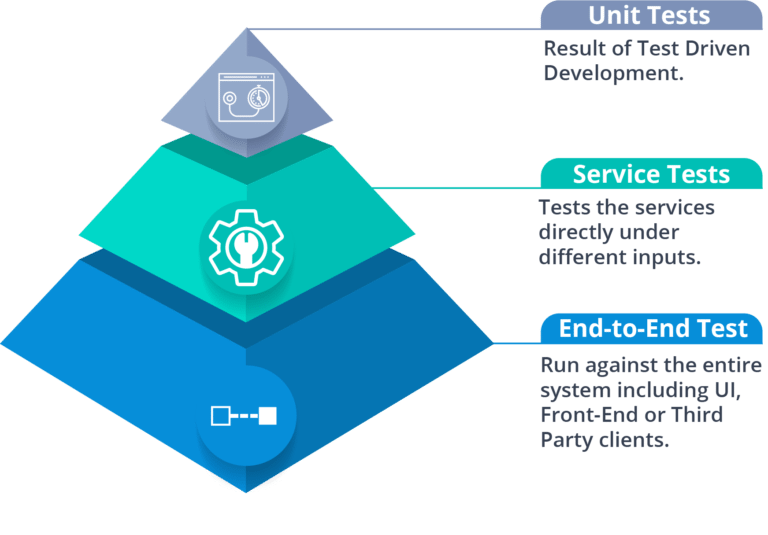
#### **Mock**

* A dummy object in which certain properties are set initially.
* The behavior of this object depends on the set properties.
* The object’s behavior can also be tested.

For example, for a Customer object, you can mock it by setting name and age. You can set age as 12 and then test for isAdult() method that will return true for age greater than 18. So, your Mock Customer object works for the specified condition.

### **Q43. What do you know about Mike Cohn’s Test Pyramid?**

**Mike Cohn** provided a model called **Test Pyramid.** This describes the kind of automated tests required for software development.



**Fig 16:**Mike Cohn’s Test Pyramid – Microservices Interview Questions

As per pyramid, the number of tests at first layer should be highest. At service layer, the number of tests should be less than at the unit test level, but more than at the end-to-end level.

### **Q44. What is the purpose of Docker?**

**Docker** provides a container environment that can be used to host any application. In this, the software application and the dependencies which support it are tightly-packaged together.

So, this packaged product is called a **Container** and since it is done by Docker, it is called **Docker container!**

[[](https://www.edureka.co/microservices-architecture-training)](https://www.edureka.co/microservices-architecture-training" \t "_blank)

### [Microservices Architecture Training](https://www.edureka.co/microservices-architecture-training" \t "_blank)

[Weekday / Weekend BatchesSee Batch Details](https://www.edureka.co/microservices-architecture-training" \t "_blank)

### **Q45. What is Canary Releasing?**

**Canary Releasing** is a technique to reduce the risk of introducing a new software version in production. This is done by slowly rolling out the change to a small subset of users before giving it out to the entire infrastructure, i.e. making it available to everybody.

### Q46. What do you mean by Continuous Integration (CI)?

**Continuous Integration (CI)** is the process of automating the build and testing of code every time a team member commits changes to version control. This encourages developers to share code and unit tests by merging the changes into a shared version control repository after every small task completion.

### **Q47. What is Continuous Monitoring?**

**Continuous monitoring** gets into the depth of monitoring coverage, from in-browser front-end performance metrics, through application performance, and down to host virtualized infrastructure metrics.

### **Q48. What is the role of an architect in Microservices architecture?**

An architect in microservices architecture plays the following roles:

* Decides broad strokes about the layout of the overall software system.
* Helps in deciding the zoning of the components. So, they make sure components are mutually cohesive, but not tightly coupled.
* Code with developers and learn the challenges faced in day-to-day life.
* Make recommendations for certain tools and technologies to the team developing microservices.
* Provide technical governance so that the teams in their technical development follow principles of Microservice.

### **Q49. Can we create State Machines out of Microservices?**

As we know that each Microservice owning its own database is an independently deployable program unit, this, in turn, lets us create a State Machine out of it. So, we can specify different states and events for a particular microservice.

For Example, we can define an Order microservice. An Order can have different states. The transitions of Order states can be independent events in the Order microservice.

### Q50. What are Reactive Extensions in Microservices?

Reactive Extensions also are known as Rx. It is a design approach in which we collect results by calling multiple services and then compile a combined response. These calls can be synchronous or asynchronous, blocking or non-blocking. Rx is a very popular tool in distributed systems which works opposite to legacy flows.

Hope these Microservices Interview Questions would help you in your Microservices Architect Interviews.

If you wish to learn Microservices and build your own applications, then check out our [***Microservices Architecture Training***](https://www.edureka.co/microservices-architecture-training) which comes with instructor-led live training and real-life project experience. This training will help you understand Microservices in depth and help you achieve mastery over the subject.

**Goal Setting**

**Goal Name:** Learn and Implement Microservices

**Description:**

Microservices is a form of service-oriented architecture style wherein applications are built as a collection of different smaller services rather than one whole app. Big and complicated applications can be made up of simpler and independent programs that are executable by themselves. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data.

**Success Criteria:**

1. Micro services Introduction and its architecture.

2. Challenges while working with Microservice.

3. Features of microservices.

4.What is Spring cloud and problems are solved by Spring Cloud?

5.Building a Microservices Application Using Spring Boot.

6. Building Architecture for an Application using Eureka Service and Discovery.

7.End to End Microservices Testing.